MemForC: Memory Forensics Corpus Creation for Malware Analysis

Augustine Orgah[1], Golden Richard III PhD[1], Andrew Case[2]
[1]Department of Computer Science, Louisiana State University, Baton Rouge, USA
[2]Department of Research, Volexity, Reston, USA
aorgah1@lsu.edu
golden@cct.lsu.edu
andrew@dfir.org

Abstract: Millions of malware samples are recorded daily with little or no information about their activity or behavior. As of 2019, recorded malware passed the billion mark according to a report from AV-Test from July 26, 2020. VirusTotal, an online malware scanner, reported over 1.4 million distinct new malware samples for seven days and over 2.3 million submissions on July 22, 2020 alone. As the arms race between malware authors and security professionals continues, it is imperative that we have better methods for detecting malware and for gaining better insight into malware behaviors. Memory forensics has emerged as a very promising set of techniques for detecting malware and analyzing malicious behavior. Memory forensics techniques can be used to detect code injection, hooks that malware places to monitor system activities, persistence mechanisms, and much more. There is, however, a critical need for ground truth data for both memory forensics investigations and to support new research in the area. For investigators, ground truth is essential in distinguishing "normal" from "malicious". For researchers, memory forensics frameworks must carefully model important data structures and algorithms, which is both difficult and frequently dependent on specific versions of operating systems and applications. Ground truth provides essential data to support testing and verification. Currently, there are no large-scale repositories that provide "known clean" memory captures for investigators to compare against those from potentially infected systems nor for developers to confirm their tools work correctly. Development of a large-scale, freely available, repository of memory captures is therefore crucial. MemForC is an open-source framework of techniques designed to create a corpus of memory captures from the successful execution of malware in Windows, Linux, and MacOS systems. MemForC is designed using best practices for creating a dynamic analysis system and leverages existing memory forensic tools. This repository will provide ground truth for investigators, allow malware research: to proceed quickly, be reproduceable and verifiable, enhance education and training to meet the demand for skilled memory forensics professionals. Our corpus will be freely available to the forensics community.

Keywords: Memory Forensics, Malware Analysis, Volatility, Dynamic Analysis, Corpus Creation

1. Introduction

The battle between security professionals and malware authors continues (Vasilescu, Gheorghe, Tapus, 2014; Ugarte-Pedrero, Graziano, Balzarotti, 2019) with malware authors having the upper hand. Miller et al (2017) found that in five years, new malware samples per year had risen from just under 100 million in 2012 to over 500 million in 2016. A report from AV-Test as of December 2, 2019 shows that new malware has continued to increase annually with over 700 million new samples in 2017 to over 980 million in 2019 and still counting (AV-Test, 2019). VirusTotal (2019), an online malware scanner, reported over one million distinct malware samples for seven days from November 24–30, 2019. There were over 2.3 million submissions on July 22, 2020 alone and over 1.4 million distinct new malware samples for seven days from July 20-30, 2020 (VirusTotal, 2020).

As malware threats rapidly grow, traditional signature-based detection schemes are no longer sufficient for detecting new variants, particularly highly targeted malware (Mosli et al., 2016). Typically, a signature for a malware sample is created manually after a security professional analyzes the sample to discover the threat(s) it poses to users (Egele et al., 2008). Despite the growing awareness of the threats posed by malware, there is still a wide gap in technological response, frameworks, tools and support to mitigate against malware and its authors (Harichandran et al., 2016). The only way that security professionals will be able to respond to the enormous amount of malware that is being developed will be by using more sophisticated and automated tools (Miller et al., 2017). The ability to accurately and efficiently analyze samples at scale will provide interesting insights and trends about malware evolution (Ugarte-Pedrero, Graziano, Balzarotti, 2019). Using static analysis (Dolly et al., 2014; Vasilescu, Gheorghe, Tapus, 2014), which are manual methods such as disassembly or reverse engineering to combat malicious software, will ultimately be impractical given the volume of malware samples generated daily (Willems, Holz, Freiling, 2007). Therefore, analysis tools must analyze malware automatically, correctly, and effectively. Dynamic analysis schemes can execute and observe malware in a controlled environment to determine what actions it performs (Dolly et al., 2014; Vasilescu, Gheorghe, Tapus, 2014; Willems, Holz, Freiling, 2007). The results obtained can also be compared to uninfected systems to better understand the behavior of the malware. Dynamic analysis also allows other useful data to be created, particularly dumps of physical memory, which can be further analyzed by existing memory forensic tools such as Volatility (2019). This is of particular importance, as memory forensics offers numerous advantages over traditional malware analysis methods. Importantly, memory "has a high potential to contain malicious code from an infection, in whole or in part, even if it's never written to disk, because it must be loaded in memory to execute" (Ligh et al., 2014).

1.1 Challenges and Motivation

To increase the breadth, scope, and accuracy of memory forensics tools to support development and testing of new memory forensics techniques, and for training purposes, there is a critical need for substantial, realistic datasets with associated ground truth. There are several important issues that will be addressed by the creation of a large corpus of memory images:

- There are restrictions on forensic research due to the lack of freely available, standardized datasets (Garfinkel et al., 2009). Often, researchers spend time generating memory datasets for memory forensics (MF) research. For example, Case and Richard III (2015) generated memory samples for their OS X rootkit detection research due to the lack of variety in available memory samples for macOS memory forensics research.

- Reproducibility continues to be the bane of digital forensics and memory forensics. Reproducibility is a key factor in scientific research and the way to validate results produced by other researchers (Garfinkel et al., 2009). Reproducibility in forensics research has been difficult due to lack of quality datasets, small sizes of the datasets, and a reluctance to share the datasets, as they are often created in an ad-hoc fashion that might leak private information.

- Lack of maintenance/updates to dataset repositories for digital forensics (DF), such as the popular Digital Corpora (digitalcorpora.org) – one of the first free online dataset repositories for digital forensics that is no longer maintained (Grajeda, Breitinger, Baggili, 2017).

Since realistic datasets are often non-existent or unsuitable for education, training, and research purposes due to the presence of information that is confidential (Woods et al., 2011), we attempt to bridge that gap with MemForC. The major contribution of this paper is to overcome the lack of realistic datasets for memory forensics and to provide a large corpus that does not include privacy-sensitive information. Furthermore, it can be freely shared to push the state of the art without concern for either privacy rights or copyright (Woods et al., 2011). This paper introduces MemForC – Memory Forensics Corpus Creation for Malware Analysis, a framework for creating a corpus of memory samples that utilizes best practices culled from existing research for malware analysis, tools, and data curation. MemForC will automatically infect machines using the Cuckoo Sandbox (Cuckoo, 2019) and produce associated uninfected memory dumps (ground truth), copies of infected memory dumps, and a comparative malware analysis contrasting the findings of Cuckoo and memory forensics tools such as Volatility. The contribution may be summarized as follows:

1) We create a corpus of memory dumps associated with dynamically analyzed malware.

2) In addition to the memory dumps, provide a comparative analysis between results from the de-facto memory forensics framework, Volatility, and Cuckoo Sandbox – a commonly used dynamic analysis framework. The research questions for this project include:

    - What are the best ways and tools to obtain and store behavioral information from malware of all types and variants effectively?

- What is the accuracy and efficiency of MemForC for malware analysis?

The rest of this paper is organized as follows: Section 2 presents an overview of related work. Section 3 introduces our research methodology. Section 4 presents the results and discussion. Section 5 discusses possibilities for future work and concludes the paper.

2. Background/Related Work

According to Garfinkel, Farrell, Roussev, and Dinot (2009), DF focuses on data extraction and evidence for presentation in court. Content is copied from a hard drive to a disk image file. The disk image file is searched for document files that are then presented as evidence to an examiner (Garfinkel et al., 2009). Memory analysis has its roots in the early 2000s, when digital forensics investigators realized that they could acquire memory directly from a running system through previously available interfaces (Case, Richard III, 2017). MF is an area of DF that provides "unprecedented visibility to the runtime state of a system" (Ligh et al., 2014). This means access to running processes, open network connections, recently executed commands, disk encryption keys, etc. This is possible because each function performed on an operating system or application makes changes to a computer's RAM, thereby preserving it. The contents of RAM supports reconstruction of events that took place, essentially providing strong correlation for traditional forensics artifacts (Ligh et al., 2014; Vasilescu et al., 2014).

Grajeda, Breitinger, and Baggili (2017) detailed the current state of affairs of datasets in the DF and MF domains. They address the importance of available datasets for MF, document the origin, kind, and availability, and what is missing from the datasets located via web searches. Their research finds that 56.4% of datasets are experimentally generated, while 36.7% are real world data. Only 3.8% of the academic articles they covered from 2010-2015 released their datasets. They have culled working locations of different datasets for DF that are updated regularly via the following site: *https://datasets.fbreitinger.de/.* This illustrates the serious lack of real datasets for MF research, education, and training.

Woods et al. discuss the lack of existing constructed realistic corpora that mimic real data without associated privacy and security concerns. They contribute to the solution by their creation and distribution of more than 40 digital forensic images, packet dumps, and memory images. These sets are free of privacy-sensitive information and can be utilized without IRB approval, and are freely redistributable without concern for either privacy rights or copyright (Woods et al., 2011). Mirroring a solution to these concerns is fundamental to the design of MemForC. Miller et al. (2017) highlight best practices and insights from constructing a scalable dynamic analysis platform, which could be used by digital forensics examiners to respond to the sheer amount of malware being developed annually. The authors utilize Cuckoo sandbox for dynamic analysis, spoofed internet access via INetSim, and provided a host of lessons they learned along the way (Miller et al., 2017). The work of the authors focuses primarily on experiments to improve configuration settings of the Cuckoo sandbox to process malware samples quickly and accurately. MemForC's primary aim is to create a corpus of memory dumps of infected machines. MemForC does not utilize Cuckoo for the memory dump creation process. MemForC utilizes a stand-alone virtual machine (VM) to execute a sample and capture a memory dump while leveraging the useful lessons learned by Miller et al. (2017) for improving dynamic analysis. Harichandran et al., (2016) highlight the general public concern about support lagging behind the escalating rate of malware authorship, the slow technological response, slow development of analysis frameworks and tools. They highlight the top five issues in computer forensics: education/training/certification, technologies, encryption, data acquisition, and tools. The authors highlight that education/training/certification and technology for cyber forensics hold the highest opportunities for growth. MemForC aims to meet those opportunities for growth by providing a framework that works efficiently, accurately, and is free. The much-needed corpus created by MemForC has the potential to substantially improve training, education, and research, as realistic datasets will be readily and freely available. As we determine which sandbox platform will suit our purposes, we consider the platform's open-source capabilities, active development, maintenance, and it meeting the requirements of a good sandbox: visibility, resistance to detection, and scalability, according to Kruegel (2014). No sandbox will ever be fully resistant to detection as malware authors continue to provide variants and updates to evade tools such as sandboxes and anti-virus software (Lindorfer et al., 2012). It should be noted that sandbox evasion techniques by malware are well documented (Lindorfer et al., 2011a; Lindorfer et al., 2012b; Ugarte-Pedrero, Graziano, Balzarotti, 2019). To analyze, extract and store behaviorial artifacts or information about malware, a sandbox strategy that offers the best possible results is our pursuit for

MemForC. Below are other sandbox systems also highlighted by Miller et al. (2017) and their current status: Anubis – web-based and currently offline; CWSandbox – commercial; Norman Sandbox – commercial and now part of AVG Technologies (AVG, 2019); Joebox (Joe-Security, 2019) – commercial. Other systems, such as Ether, WILDCat, Panorama, TEMU, have either no source code or are not maintained. Cuckoo sandbox was chosen because it is open-source, robust, and meets other criteria for a good sandbox. It also is very popular in the malware analysis community (Miller et al., 2017). Volatility is popular within the incident response and malware analysis communities and has the most robust support for Windows and other operating systems, especially for malware detection and analysis (Case, Richard III, 2015). We thus utilize Volatility as part of our framework MemForC.

3. Research Methodology

To create realistic datasets for memory forensics, memory dumps and accurate behaviorial data must be captured. We have to provide all the resources that malware needs to run and the semblance of a 'real' system to deter the malware from employing anti-analysis antics during execution. If the malware obfuscates its behavior or refuses to run then its behavior eludes capture of artifacts in RAM and associated analysis. To ensure that malware reveals its true behavior for capture and analysis, we employ several techniques such as anti-VM detection, VM-hardening, anti-sandbox detection and specific system configurations to improve malware execution. This section describes our proposed framework MemForC. It consists of several modules: infection module with Cuckoo Sandbox and VMware; the memory analysis module with Volatility; and the comparative analysis module. At the end of a run, memory dumps are created from the infection module for the corpus and result of the comparative analysis module identifies the behavior of the malware sample based on the following artifacts and interactions: APIs, files, registry, mutexes, and network. Figure. 1 provides an overview of MemForC.

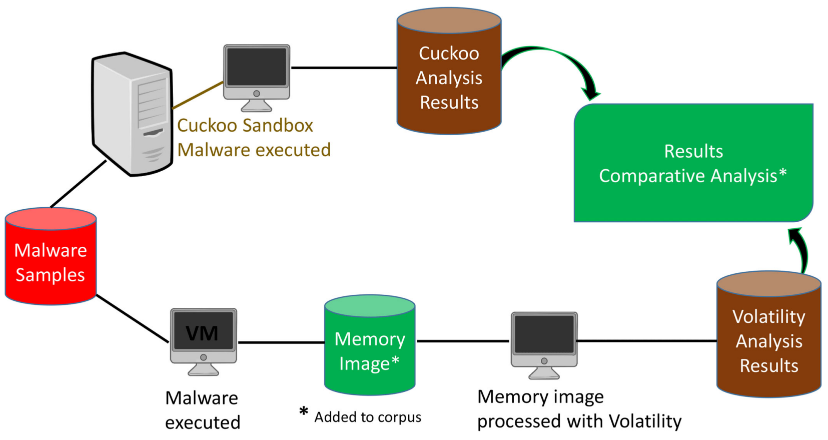

Cuckoo Analysis Results
Cuckoo Sandbox Malware executed
Results Comparative Analysis*
Malware Samples
VM
Malware executed
Memory Image*
* Added to corpus
Memory image processed with Volatility
Volatility Analysis Results

Figure 1: MemForC Architecture

3.1 Module 1 - Infection & Execution

3.1.1 Cuckoo Infection

MemForC currently utilizes an Ubuntu 18.04 LTS 64-bit system with 64GB of RAM and 12 cores as the host in a bidirectional setup with the analysis and agent VMs. Figure 2 (Cuckoo, 2019) illustrates the setup we utilized. VMware is our virtualization software of choice and VMware tools are intentionally not installed for Cuckoo analyses due to footprint and anti-VM detection consequences. We employ this strategy since malware can terminate execution or provide a different behavior if it detects it is being run in a VM. The Cuckoo host allows the submission of files to be run in an isolated environment in a VM. A sample is sent to the agent VM from the host machine to be executed. Cuckoo first reverts the VM to a base snapshot prepared previously and not infected by malware. It then executes the malware in a VM (Miller et al., 2017; Cuckoo, 2019) that is hardened to obfuscate tell-take signs of virtualization. As the sample executes, the Cuckoo host (our Ubuntu system) collects information about the behavior of the malware on the host from categories such as: API calls, network traffic, files, mutexes, and registry keys accessed.

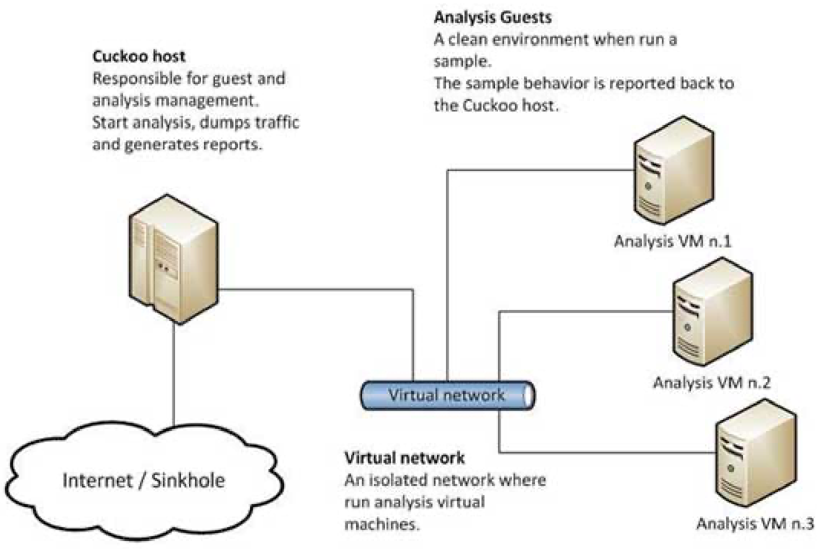

Analysis Guests
A clean environment when run a sample.
The sample behavior is reported back to the Cuckoo host.
Cuckoo host
Responsible for guest and analysis management.
Start analysis, dumps traffic and generates reports.
Analysis VM n.1
Analysis VM n.2
Virtual network
Internet / Sinkhole
Virtual network
An isolated network where run analysis virtual machines.
Analysis VM n.3

Figure 2: Cuckoo (Cuckoo, 2019) setup utilized for MemForC

We are currently concentrating our efforts primarily on Windows malware. The agent VM - the system infected with malware - has the following characteristics: Windows 7 SP1 32-bit, 7GB of RAM, and 2 CPU cores. Installed are: AdobeReader DC, Adobe Flash 11, browsers: Chrome 78 and Firefox 66.01, 7zip, Python 2.7, Python PIL 1.1.7, Microsoft .Net Framework 4.61, and Visual C++: 2005, 2008, 2010, 2012, 2013, and 2015. Disabled are: Window User Account Control (UAC), firewall, auto updates, Microsoft malicious software removal tool and no antivirus was installed. To harden the agent, the following actions are taken:

- Remove/replace registry traces of VM software
- No installed VMware Tools
- Change VM MAC address and network interfaces from default
- Add/delete files, change desktop background, and have system appear to be in use by actual user(s)
- Before taking a snapshot, VM uptime set at least 30 minutes to avoid failure of tests for VM or sandbox evasion
- Ensure that our VM has sufficient resources and speed to prevent malware anti-analysis due to the VM being slower than a real system (Lindorfer et al., 2011)

We validate our VM-hardening and anti-VM detection using the Paranoid Fish (Ortega, 2019) – a tool for identifying sandboxes, analysis environments and common malware techniques to fingerprint systems (Miller et al., 2017). Although no system can ever be fully hardened against VM detection, we apply our best knowledge and de-facto techniques for hardening.

3.1.2 VM-Only Infection

In this section, instead of utilizing Cuckoo as the environment for execution of malware and obtaining behavioral info, we utilize a Windows VM with almost the same machine specifications as the Cuckoo agent VM. We install VMWare Tools as it is necessary for submission and enabling execution of our samples in our solution. It is also hardened in a similar manner without losing functionality. We keep a copy of this VM that is uninfected as ground truth. We created a script (Ligh et al., 2010) that copies the malware from a location on the host to the VM. It executes the malware for 30 seconds and then suspends the VM to create a memory dump. This memory dump is added to storage, building our corpus. This memory dump is slated for further analysis to extract interactions and behavior in Module 2 - MF analysis.

3.2 Module 2 - Memory Forensics

At this stage of analysis, we are equipped with a memory image from the execution of the malware sample in the VM. We process this memory dump with the popular MF framework, Volatility, to obtain behavioral info about the malware sample.

We currently utilize the following Volatility plugins (Volatility, 2019):

- Atomscan - Obtain atom tables
- Filescan - Obtain file objects
- Hivelist - Obtain registry hives
- Printkey - Obtain a registry key, its subkeys and values
- Mutantscan - Obtain mutex objects
- Netscan - Obtain connections and sockets

The results obtained from these plugins will be utilized in a comparative analysis with the behavioral information obtained from Module 1 - Cuckoo infection & execution.

3.3 Module 3 - Comparative Analysis

This is the final processing module of MemForC. This module will provide results about the behavior of a malware sample based on analysis from Volatility and Cuckoo Sandbox. Specifically, the results from the analysis via Cuckoo (Module 1 – Cuckoo infection) are compared for similarity with the results in Module 2 – MF with Volatility. An API listing from the Microsoft Development Center (Microsoft, 2019) is culled for the following API categories:

- Filenames - fileapi.h
- Mutexes - synchapi.h
- Network/IP - winsock2.h & ws2tcpip.h
- Registry - winreg.h

The culled API call functions are searched for within the results from the Cuckoo analysis for each category. These categories are compared for similarities with those results obtained from the results obtained via Volatility analysis in Module 2. For instance, are there common file APIs and files found via Module 2 and Module 1 – Cuckoo infection results? Similar findings reveal that platforms (Cuckoo and Volatility) found similar artifacts and that the behavior of the malware was replicated in both scenarios. If the comparative processing reveals dissimilarities, then it could mean that the behavior of the malware was dissimilar in both scenarios and its true behavior was revealed in one scenario but not the other. This would require further research and reviewing the result files and logs to determine the possible reasons. The results of this phase of MemForC are stored in the corpus along with the memory dump.

4. Preliminary/Experimental Results and Discussion

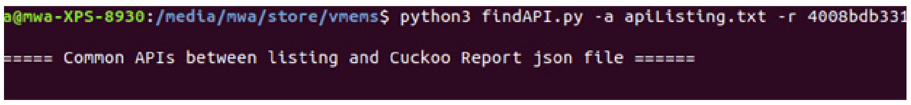
We performed preliminary tests of MemForC with seven samples, all Windows executables. One of the executables was obtained from the GitHub repository, theZoo (theZoo, 2019) and the others were randomly selected from our own large corpus of malware samples to be analyzed via MemForC. The summary of results, as illustrated in Table 1, are promising, as the samples ran successfully and with our scripts we were able to provide comparative results for files, network and registry artifacts. We executed all samples in a private network without an internet connection as a safety measure. It is well established that malware will morph its behavior, stay dormant, or maybe even not execute at all if no network is detected. We are aware that this as well as other anti-analysis techniques by malware will affect the veracity of the results obtained.

We executed each of these samples via MemForC and Cuckoo Sandbox and compared results but importantly, also saved a memory dump from the execution of each sample when executed through the MemForC framework. Figures 3-7 illustrate some of the results obtained from execution of malware samples via the MemForC framework.

Analysis Summary						
Sample		Malware Sample	Sandbox Execution			
ID	Type	Name	Cuckoo	Results	MemForC	Results
1	EXE	vKazy.exe[1]	Yes	Yes	Yes	Yes
2	PE32 EXE	10546d23653_a60a5375b9	Yes	Yes	Yes	Yes
3	PE32 EXE	24f133fc78f_209dbd524c	Yes	Yes	Yes	Yes
4	PE32 EXE	24f89cad806_b2a7f45fef	Yes	Yes	Yes	Yes
5	PE32 EXE	4008bdb3310_85c1da0d11	Yes	Mixed	Yes	Yes
6	PE32 EXE	960a6d242ac_50fa9da858	Yes	Mixed	Yes	Yes
7	PE32 EXE	cbb12f29999_d82f1e5bde	Yes	Yes	Yes	Yes

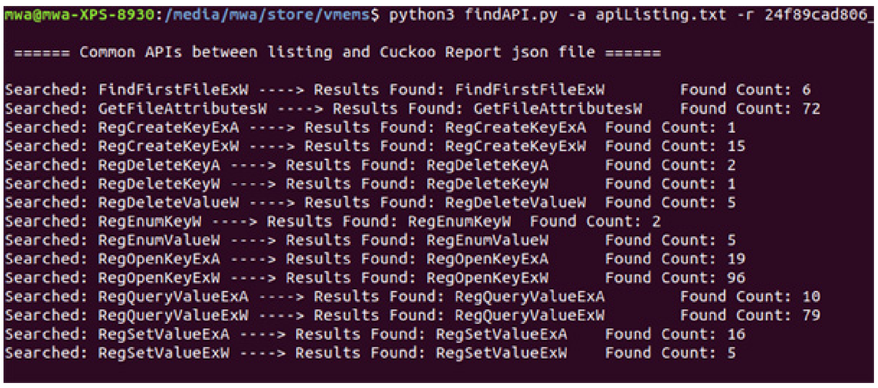
Table 1: Summary Comparative Analysis. [1]Sample from theZoo (theZoo, 2019), others taken randomly from corpus of malware samples available to our lab and names shortened for brevity.

From Table 1, although all samples executed, samples 5 and 6 are labelled as mixed, because when executed using Cuckoo, they both turned up no APIs from our targeted list. This issue is currently under investigation. Figure 3 shows the result with sample 5 and Figure 4 shows successful results using sample 4.



```
a@mwa-XPS-8930:/media/mwa/store/vmems$ python3 findAPI.py -a apiListing.txt -r 4008bdb331

===== Common APIs between listing and Cuckoo Report json file ======
```

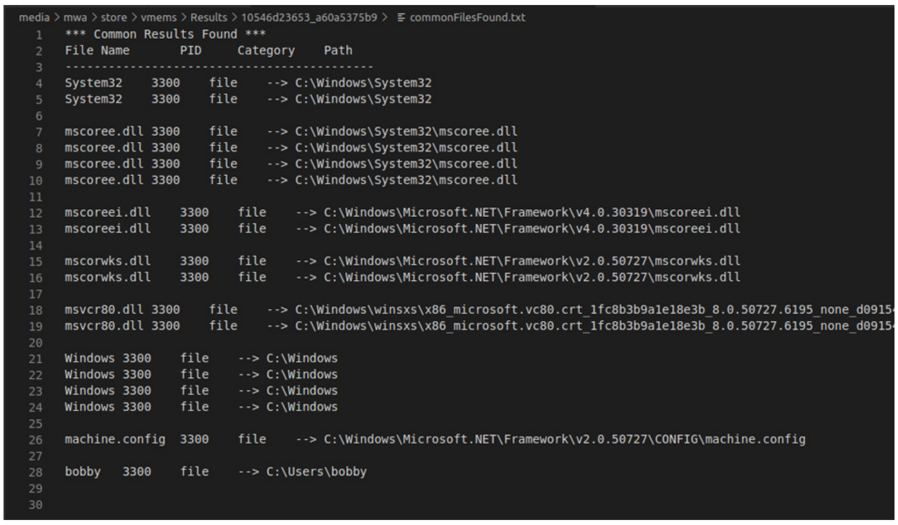
Figure 3: Sample 5 Comparative API listings between MemForC and Cuckoo (Cuckoo, 2019)



```
mwa@mwa-XPS-8930:/media/mwa/store/vmems$ python3 findAPI.py -a apiListing.txt -r 24f89cad806_

 ====== Common APIs between listing and Cuckoo Report json file ======

Searched: FindFirstFileExW ----> Results Found: FindFirstFileExW        Found Count: 6
Searched: GetFileAttributesW ----> Results Found: GetFileAttributesW    Found Count: 72
Searched: RegCreateKeyExA ----> Results Found: RegCreateKeyExA  Found Count: 1
Searched: RegCreateKeyExW ----> Results Found: RegCreateKeyExW  Found Count: 15
Searched: RegDeleteKeyA ----> Results Found: RegDeleteKeyA      Found Count: 2
Searched: RegDeleteKeyW ----> Results Found: RegDeleteKeyW      Found Count: 1
Searched: RegDeleteValueW ----> Results Found: RegDeleteValueW  Found Count: 5
Searched: RegEnumKeyW ----> Results Found: RegEnumKeyW  Found Count: 2
Searched: RegEnumValueW ----> Results Found: RegEnumValueW      Found Count: 5
Searched: RegOpenKeyExA ----> Results Found: RegOpenKeyExA      Found Count: 19
Searched: RegOpenKeyExW ----> Results Found: RegOpenKeyExW      Found Count: 96
Searched: RegQueryValueExA ----> Results Found: RegQueryValueExA        Found Count: 10
Searched: RegQueryValueExW ----> Results Found: RegQueryValueExW        Found Count: 79
Searched: RegSetValueExA ----> Results Found: RegSetValueExA    Found Count: 16
Searched: RegSetValueExW ----> Results Found: RegSetValueExW    Found Count: 5
```

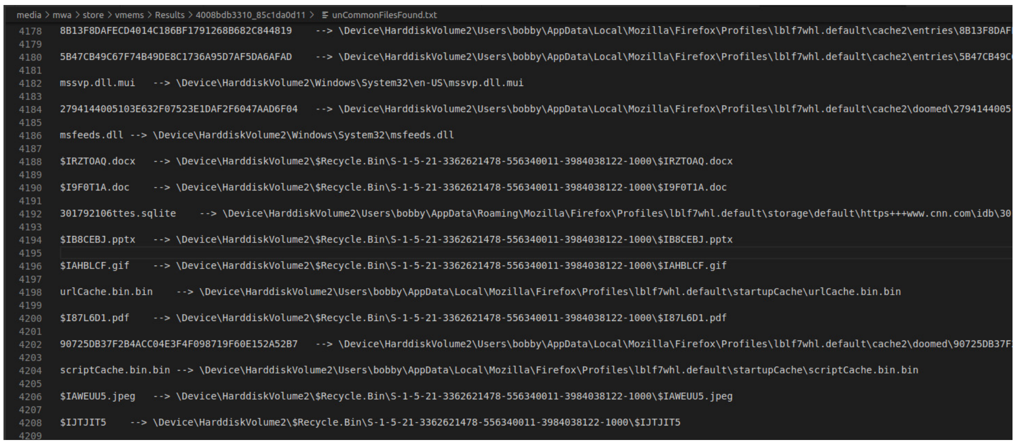
Figure 4: Sample 4 Comparative API listings between MemForC and Cuckoo (Cuckoo, 2019)

Samples 5 and 6 provide limited data from their execution via Cuckoo which in turn leads to limited comparable artifacts from the results obtained by MemForC. Figure 6 for instance illustrates file artifacts found by MemForC but not identified via execution in Cuckoo.



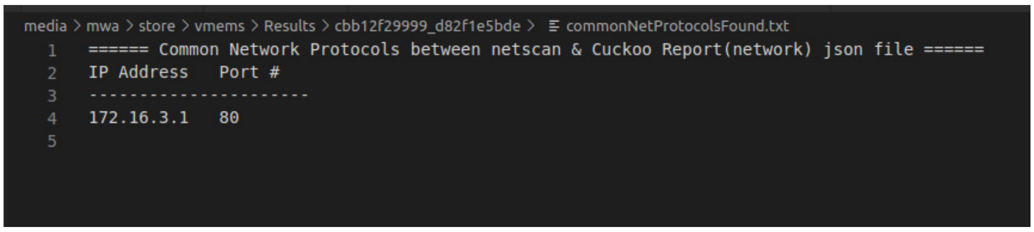
```
media > mwa > store > vmems > Results > 10546d23653_a60a5375b9 > commonFilesFound.txt
*** Common Results Found ***
File Name       PID     Category    Path
-------------------------------------------
System32    3300    file    --> C:\Windows\System32
System32    3300    file    --> C:\Windows\System32

mscoree.dll 3300    file    --> C:\Windows\System32\mscoree.dll
mscoree.dll 3300    file    --> C:\Windows\System32\mscoree.dll
mscoree.dll 3300    file    --> C:\Windows\System32\mscoree.dll
mscoree.dll 3300    file    --> C:\Windows\System32\mscoree.dll

mscoreei.dll    3300    file    --> C:\Windows\Microsoft.NET\Framework\v4.0.30319\mscoreei.dll
mscoreei.dll    3300    file    --> C:\Windows\Microsoft.NET\Framework\v4.0.30319\mscoreei.dll

mscorwks.dll    3300    file    --> C:\Windows\Microsoft.NET\Framework\v2.0.50727\mscorwks.dll
mscorwks.dll    3300    file    --> C:\Windows\Microsoft.NET\Framework\v2.0.50727\mscorwks.dll

msvcr80.dll 3300    file    --> C:\Windows\winsxs\x86_microsoft.vc80.crt_1fc8b3b9a1e18e3b_8.0.50727.6195_none_d0915
msvcr80.dll 3300    file    --> C:\Windows\winsxs\x86_microsoft.vc80.crt_1fc8b3b9a1e18e3b_8.0.50727.6195_none_d0915

Windows 3300    file    --> C:\Windows
Windows 3300    file    --> C:\Windows
Windows 3300    file    --> C:\Windows
Windows 3300    file    --> C:\Windows

machine.config  3300    file    --> C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG\machine.config

bobby   3300    file    --> C:\Users\bobby


```

Figure 5: File artifacts for Sample 2 identified by both Cuckoo (Cuckoo, 2019) and MemForC



```
media > mwa > store > vmems > Results > 4008bdb3310_85c1da0d11 > unCommonFilesFound.txt
4178  8B13F8DAFECD4014C186BF1791268B682C844819    --> \Device\HarddiskVolume2\Users\bobby\AppData\Local\Mozilla\Firefox\Profiles\lblf7whl.default\cache2\entries\8B13F8DAF
4179
4180  5B47CB49C67F74B49DE8C1736A95D7AF5DA6AFAD    --> \Device\HarddiskVolume2\Users\bobby\AppData\Local\Mozilla\Firefox\Profiles\lblf7whl.default\cache2\entries\5B47CB49C
4181
4182  mssvp.dll.mui   --> \Device\HarddiskVolume2\Windows\System32\en-US\mssvp.dll.mui
4183
4184  2794144005103E632F07523E1DAF2F6047AAD6F04   --> \Device\HarddiskVolume2\Users\bobby\AppData\Local\Mozilla\Firefox\Profiles\lblf7whl.default\cache2\doomed\2794144005
4185
4186  msfeeds.dll --> \Device\HarddiskVolume2\Windows\System32\msfeeds.dll
4187
4188  $IRZTOAQ.docx   --> \Device\HarddiskVolume2\$Recycle.Bin\S-1-5-21-3362621478-556340011-3984038122-1000\$IRZTOAQ.docx
4189
4190  $I9F0T1A.doc    --> \Device\HarddiskVolume2\$Recycle.Bin\S-1-5-21-3362621478-556340011-3984038122-1000\$I9F0T1A.doc
4191
4192  301792106ttes.sqlite    --> \Device\HarddiskVolume2\Users\bobby\AppData\Roaming\Mozilla\Firefox\Profiles\lblf7whl.default\storage\default\https+++www.cnn.com\idb\30
4193
4194  $IB8CEBJ.pptx   --> \Device\HarddiskVolume2\$Recycle.Bin\S-1-5-21-3362621478-556340011-3984038122-1000\$IB8CEBJ.pptx
4195
4196  $IAHBLCF.gif    --> \Device\HarddiskVolume2\$Recycle.Bin\S-1-5-21-3362621478-556340011-3984038122-1000\$IAHBLCF.gif
4197
4198  urlCache.bin.bin    --> \Device\HarddiskVolume2\Users\bobby\AppData\Local\Mozilla\Firefox\Profiles\lblf7whl.default\startupCache\urlCache.bin.bin
4199
4200  $I87L6D1.pdf    --> \Device\HarddiskVolume2\$Recycle.Bin\S-1-5-21-3362621478-556340011-3984038122-1000\$I87L6D1.pdf
4201
4202  90725DB37F2B4ACC04E3F4F098719F60E152A52B7   --> \Device\HarddiskVolume2\Users\bobby\AppData\Local\Mozilla\Firefox\Profiles\lblf7whl.default\cache2\doomed\90725DB37F
4203
4204  scriptCache.bin.bin --> \Device\HarddiskVolume2\Users\bobby\AppData\Local\Mozilla\Firefox\Profiles\lblf7whl.default\startupCache\scriptCache.bin.bin
4205
4206  $IAWEUU5.jpeg   --> \Device\HarddiskVolume2\$Recycle.Bin\S-1-5-21-3362621478-556340011-3984038122-1000\$IAWEUU5.jpeg
4207
4208  $IJTJIT5    --> \Device\HarddiskVolume2\$Recycle.Bin\S-1-5-21-3362621478-556340011-3984038122-1000\$IJTJIT5
4209
```

Figure 6: MemForC file interactions not identified by Cuckoo (Cuckoo, 2019)



```
media > mwa > store > vmems > Results > cbb12f29999_d82f1e5bde > commonNetProtocolsFound.txt
====== Common Network Protocols between netscan & Cuckoo Report(network) json file ======
IP Address    Port #
----------------------
172.16.3.1   80

```

Figure 7: Common network artifacts identified by both Cuckoo (Cuckoo, 2019) and MemForC

We acknowledge that many more samples need to be analyzed to validate and verify MemForC's accuracy and robustness. We currently are continuing to improve MemForC as we evaluate new malware samples, to increase speed, efficiency, and storage space, as our ultimate goal is to create a very large repository.

5. Conclusion and Future Work

In this paper we present MemForC, a framework to create a corpus of memory dumps that have been successfully infected with malware. We also store the associated ground truth for each malware sample that we execute on our framework. We utilize virtual machines and best practices according to Miller et al (2017) to create our framework. MemForC is designed to support filling current gaps in the memory forensics arena, by creating freely available, realistic datasets, free of personal identifiable information, which are suitable for education, training, and research.

Our future work involves optimization of the framework and finding better ways to ensure that MemForC accurately executes malware to expose its behaviour in an efficient and timely manner. We also intend to expand the analysis to include other versions of the Windows operating system and to target other operating systems.

References

AVG, 2019. AVG Technologies. [Online]. Available at: https://www.avg.com/en-us/

AV-Test, 2019. AV-Test Institute Malware. [Online]. Available at: https://www.av-test.org/en/statistics/malware/(2019/12/1) [Accessed 1 Dec. 2019].

Case, A. and Richard III, G.G., 2015. Advancing Mac OS X rootkit detection. Digital Investigation, 14, pp.S25-S33.

Case, A. and Richard III, G.G., 2017. Memory forensics: The path forward. Digital Investigation, 20, pp.23-33.

Cuckoo, 2019. Cuckoo Sandbox - Automated Dynamic Malware Analysis System. [Online]. Available at: https://cuckoosandbox.org/

Egele, M., Scholte, T., Kirda, E. and Kruegel, C. (2008) A survey on automated dynamic malware-analysis techniques and tools. ACM computing surveys (CSUR), 44(2), pp.1-42.

Garfinkel, S., Farrell, P., Roussev, V. and Dinolt, G., 2009. Bringing science to digital forensics with standardized forensic corpora. digital investigation, 6, pp.S2-S11.

Grajeda, C., Breitinger, F. and Baggili, I., 2017. Availability of datasets for digital forensics–And what is missing. Digital Investigation, 22, pp.S94-S105.

Harichandran, V.S., Breitinger, F., Baggili, I. and Marrington, A. (2016) A cyber forensics needs analysis survey: Revisiting the domain's needs a decade later. Computers & Security, 57, pp.1-13.

Joe-Security, 2019. Joe Sandbox: Automated Malware Analysis. [Online]. Available at: https://www.joesecurity.org/

Kruegel, C., 2014, August. Full system emulation: Achieving successful automated dynamic analysis of evasive malware. In Proc. BlackHat USA Security Conference (pp. 1-7).

Ligh, M., Adair, S., Hartstein, B. and Richard, M., 2010. Malware analyst's cookbook and DVD: tools and techniques for fighting malicious code. Wiley Publishing.

Ligh, M.H., Case, A., Levy, J. and Walters, A., 2014. The art of memory forensics: detecting malware and threats in windows, linux, and Mac memory. John Wiley & Sons.

Lindorfer, M., Di Federico, A., Maggi, F., Comparetti, P.M. and Zanero, S., 2012, December. Lines of malicious code: insights into the malicious software industry. In Proceedings of the 28th Annual Computer Security Applications Conference (pp. 349-358).

Lindorfer, M., Kolbitsch, C. and Comparetti, P.M., 2011, September. Detecting environment-sensitive malware. In International Workshop on Recent Advances in Intrusion Detection (pp. 338-357). Springer, Berlin, Heidelberg.

Microsoft, 2019. Microsoft Development Center. [Online]. Available at: https://docs.microsoft.com/en-us/windows/win32/api/

Miller, C., Glendowne, D., Cook, H., Thomas, D., Lanclos, C. and Pape, P. (2017) Insights gained from constructing a large scale dynamic analysis platform. Digital Investigation, 22, pp.S48-S56.

Mosli, R., Li, R., Yuan, B. and Pan, Y. (2016, May) Automated malware detection using artifacts in forensic memory images. In 2016 IEEE Symposium on Technologies for Homeland Security (HST) (pp. 1-6). IEEE.

Ortega, A., 2019. Github - Pafish (Paranoid Fish). [Online]. Available at: https://github.com/a0rtega/pafish

theZoo, 2019. Github - theZoo - A live Malware Repository. [Online]. Available at: https://github.com/ytisf/theZoo

Ugarte-Pedrero, X., Graziano, M. and Balzarotti, D. (2019) A close look at a daily dataset of malware samples. ACM Transactions on Privacy and Security (TOPS), 22(1), pp.1-30.

Uppal, D., Sinha, R., Mehra, V. and Jain, V.(2014 September) Malware detection and classification based on extraction of API sequences. In 2014 International conference on advances in computing, communications and informatics (ICACCI) (pp. 2337-2342). IEEE.

Vasilescu, M., Gheorghe, L. and Tapus, N. (2014, September) Practical malware analysis based on sandboxing. In 2014 RoEduNet Conference 13th Edition: Networking in Education and Research Joint Event RENAM 8th Conference (pp. 1-6). IEEE.

VirusTotal, 2019. Virustotal file statistics.. [Online]. Available at: https://www.virustotal.com/en/statistics/

Volatility, 2019. The Volatility Framework. [Online]. Available at: https://github.com/volatilityfoundation

Willems, C., Holz, T. and Freiling, F., 2007. Toward automated dynamic malware analysis using cwsandbox. IEEE Security & Privacy, 5(2), pp.32-39.

Woods, K., Lee, C.A., Garfinkel, S., Dittrich, D., Russell, A. and Kearton, K., 2011. Creating realistic corpora for security and forensic education. NAVAL POSTGRADUATE SCHOOL MONTEREY CA DEPT OF COMPUTER SCIENCE.